JavaRX

Andrea Marín Alarcón
158999

RxJava es un tipo de programación reactiva que nos permite trabajar con sistemas asíncronos al crear sistemas reactivos. La solución usual para manejar llamadas asíncronas en sistemas síncronos es implementar callbacks y listeners para poder controlar tanto el caso de éxito como el caso de fallo.

El problema es cuando tienes más de una llamada asíncrona pues crecen los casos que debes manejar y también se deben de tomar en cuenta los procesos que están ocurriendo de forma paralela que pueden afectar el funcionamiento de mi programa. Además de todo eso también se tienen que manejar las entradas de los usuarios y el funcionamiento básico de la aplicación como cuándo habilitar los componentes y manejar *requests* simultáneos.

En resumen tenemos que comunicarnos con la red que es una fuente asíncrona, también está la interfaz de usuario que también es una fuente asíncrona y todo esto va a estar pasando de manera concurrente y el código es el que tiene que manejar todos estos datos. También tenemos que tomar en cuenta que es necesario reaccionar a cualquier cambio que se presente.

RxJava permite quitarle al código la responsabilidad de reaccionar a todos los cambios y hace que las diferentes fuentes estén conectadas una a la otra. De esta manera, los cambios en una fuente son transmitidos inmediatamente al destino necesario; por ejemplo, cuando el usuario ingresa información esto se transmite directamente a la base de datos para que se actualice sin la necesidad de que un código lo haga. Esto se logra implementando streams de los objetos en forma de *Observables* los cuales te notifican cuando el objeto ha sufrido un cambio.

Con RxJava tenemos:

- Pushed-based updates
- Declarative threading (realizar cambios en una sola línea)
- Manjeo de errores de manera fácil
- Callbacks especializados

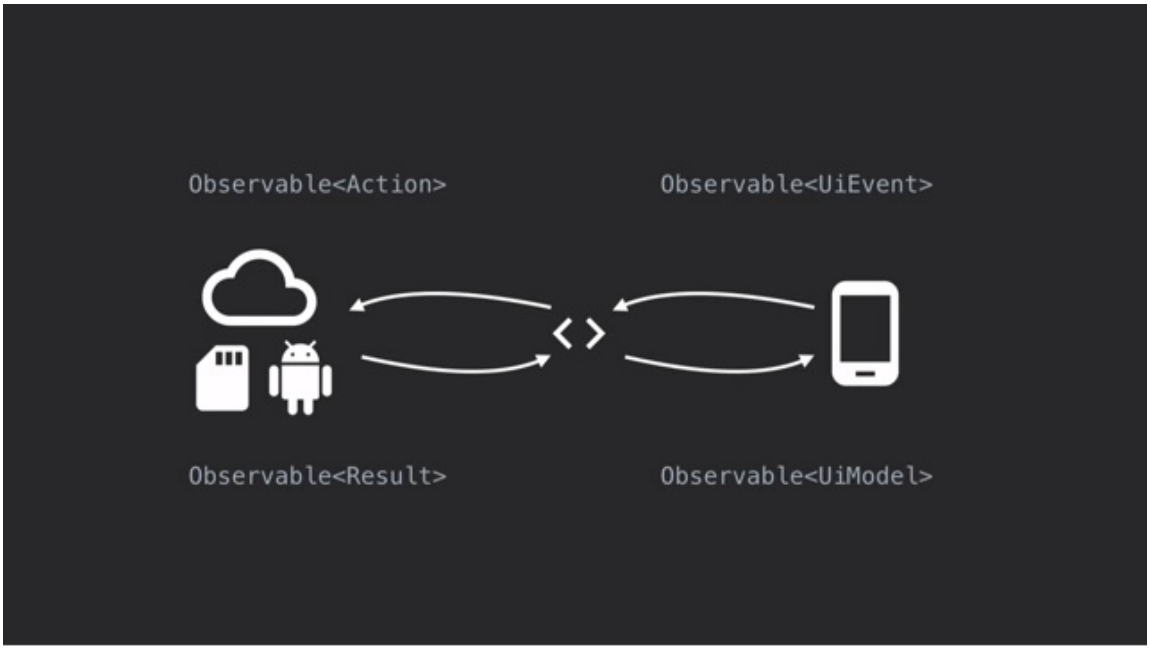
RxJava facilita la comunicación entre entidades pero aún tenemos que manejar todas las *requests* que lleguen. Una manera de manejar los *requests* de manera sencilla sin bugs que nos puedan generar problemas es que al momento que se genere un *request* el UI nos regrese un stream con toda la información necesaria para completarlo, de esta manera no tenemos que regresar a la UI para obtener los datos. Además hacemos que un stream de eventos se transmita al UI por medio del método `suscribe()`. Estos métodos nos permiten manejar los datos que se transmiten del UI al código y viceversa. Si tenemos más de un *request* proveniente del UI entonces los combinamos en un solo stream.

Para manejar el flujo de datos del código a otros componentes (red, base dadtos, sistema de archivos, entre otros) necesitamos algo que desencadene las acciones asíncronas de los componentes, así como otra cosa que represente el resultado de la acción. Por lo que el código se convierte en un traductor entre estos dos sistemas. Además necesitamos algo que rastree las actualizaciones del modelo que posteriormente se implementa en la UI.

RxJava tiene un operador que nos permite realizar una actualización incremental de los estados llamada `scan`. Se comienza con el estado base, luego para cada evento hay un *callback* que regresa un nuevo estado el cual se convierte en el nuevo estado base hasta que haya otro evento.

De esta manera no tenemos que manejar el modelo directamente si no que sólo manejamos las acciones que pasan en el segundo plano que no saben nada del modelo y que pueden ser reusadas en cualquier parte de la aplicación. Es importante aclarar que estas acciones no son lo mismo que los eventos del UI, para que de esta manera los procesos asíncronos estén ligados a la UI y se puedan manejar varios *requests*. Los modelos contienen toda la información importante de los estados que necesita ser transmitida al UI.

De esta manera el flujo de datos se ve de la siguiente manera:



Podemos ver que RxJava cambia el paradigma de programación volviendo más fácil el manejo de datos y minimizando los bugs ocasionados por mal código. Cada vez hay más sistemas que funcionan de manera asíncrona y que requieren de flujos de información complejos y con RxJava

gestionar estos procesos se vuelve mucho más sencillo.

RxJava está basado en objetos Observables los cuales definen una dependencia de uno-a-muchos entre objetos de manera que en cuanto uno de los objetos cambia su estado, este cambio se notifica a todos sus dependientes de inmediato. Esto nos evita tener que estar iterando sobre todos los objetos para ver cuál de ellos ha cambiado, sino que sólo esperas a que llegue la notificación de cambio para reaccionar como se necesite.

Hoy en día los clientes esperan que la información que obtienen esté en tiempo real. Quieren que todo ocurra en el momento en el que lo requieren. Por eso, como programador, requieres de una herramienta que te permite mostrar los resultados en cuanto estén listos y, como vimos en la tarea anterior, la programación reactiva es la mejor manera de hacerlo.